Apache Storm on HDInsight

Apache Storm on HDInsight.
Apache Storm is a system for
processing streaming data in real-time.
It is a distributed real-time computation system for
processing large volumes of high velocity data.
It is extremely fast with the ability to process
over 1 million records per second per node
on a cluster of modest size.
Apache Storm is a distributed,
fault tolerant open source computation system that
allows you to process data in real-time with Hadoop.
Storm solutions can also provide guaranteed processing of data
with the ability to replay data that was not successfully
processed the first time.
Five characteristics makes Storm ideal for
real time data processing work loads.
Storm is fast, benchmarked as processing 1 million 100 byte
messages per second, per node.
Scalable, with parallel calculations that run across
a cluster of machines.
Fault tolerant,
when worker's die, Storm will automatically restart them.
If a node dies, the worker will be restarted on another node.
Reliable, Storm guarantees that each unit of data,
of tuple, will be processed at least once or exactly once.
Messages are only replayed when there are failures.
It is easy to operate.
Standard configurations are suitable for
production on day one.
Once deployed, Storm is easy to operate.
Event Hubs allow you to process massive amounts of data from
websites, apps and devices.
The Event Hubs spout makes it easy to use Apache Storm on
HDInsight to analyze this data in real-time.
The Azure Event Hub service is a highly scalable telemetry
ingestion service that can log millions of events per second in
near real-time.
It can be used to collect data and
events from any IOT device, from any app, web, mobile,
or back-end service, or via feeds like social networks.
Once you collect events with Event Hub, you can then
analyze the data using any real-time analytics system,
like Apache Storm or our new Azure Stream Analytics service.
And store or transform it into any data storage system,
including HDInsight and Hadoop based solutions.
Event Hub is delivered as a managed service in Azure,
meaning we run scale and patch it for you and
provide an enterprise SLA.

Storm has a simple and easy to use API.
When programming on Storm, you manipulate and
transform streams of tuples, which are named lists of values.
Tuples can contain objects of any type.
If you want to use a type Storm doesn't know about,
it's very easy to register a serializer for that type.
While in Hadoop, you have MapReduce jobs,
in Storm you have two types of notes, spouts and bolts.
These nodes can receive data, tuples, process them, and
emit them to other nodes so they can be chained together.
A storm topology consists of three components.
Spouts, which read data from external sources and
emit streams of data into the topology.
Bolts, that perform processing on streams emitted by spouts or
other bolts and emit one or more streams.
The topology defines how the spouts and bolts are arranged,
and provides the entry point for the topology.
A Storm cluster is superficially similar to a Hadoop cluster.
There are two kinds of nodes on a Storm cluster,
the master node and the worker nodes.
The master node run a deamon called Nimbus,
which is similar to Hadoop's Job Tracker.
Nimbus is responsible for
distributing code around the cluster,
assigning tasks to machines, and monitoring for failures.
Each worker node runs a deamon called the Supervisor.
The Supervisor listen for work assigned to it's machine and
starts and stops worker processes as necessary,
based on what Nimbus has assigned to it.
Each worker process executes a subset of a topology.
A running topology consist of mainly worker processes spread
across many machines.
All coordination between Nimbus and
the Supervisors is done through a ZooKeeper cluster.
Additionally, the Nimbus daemon and
the Supervisor daemons are fail fast and stateless.
All state is kept in ZooKeeper or on a local disk.
This means you can kill a Nimbus or the Supervisors, and
they'll start back up like nothing happened.
This design leads to Storm clusters being
incredibly stable.
A spout is a source of streams in a topology.
Generally, spouts will read tuples from an external source
and emit them into the topology, for example,
like a cast or queue or the Twitter API.
Spouts can either be reliable or unreliable,
a reliable spout is capable of replaying a tuple if it
failed to be processed by Storm whereas an unreliable spout
forgets about the tuple as soon as it's emitted.
Spouts can emit more than one stream.

To do so, declare multiple streams using the declare stream
method of output fields declarer and specify the stream to emit
to when using the emit method on spout output collector.
The main method on spouts is, nextTuple.
NextTuple either emits a new tuple into the topology, or
simply returns it, if there are no new tuples to emit.
It is imperative that nextTuple does not block for
any spout implementation, because Storm calls
all the spout methods on the same thread.
The other main methods on spouts are act and fail.
These are called when storm detects that a tuple emitted
from the spout either successfully completed through
the topology or failed to be completed.
Act and fail are only called for reliable spouts.
All processing in topologies is done in bolts.
Bolts can do anything from filtering, functions,
aggregations, joins, talking to databases, and more.
Bolts can do simple stream transformations.
Doing complex stream transformations often requires
multiple steps, and thus multiple bolts.
Bolts can emit more than one stream.
To do so, declare multiple streams using the declare stream
method of output fields declare, and specify the stream to
emit to when using the emit method on output collector.
When you declare a bolt's input stream,
you always subscribe to specific streams of another component.
If you want to subscribe to all the streams of another
component, you have to subscribe to each one individually.
The main method in bolts is the execute method
which takes a new tuple as an input.
Bolt emit new tuples using the output collector object.
Bolts must call the act method on the output collector for
every tuple they process so
that is Storm knows when tuples are completed.
For the common case of the processing an input tuple
emitting zero or more tuples based on that tuple and
then hacking the input tuple, Storm provides an iBasic bolt
interface which does the hacking automatically.
It's perfectly fine to launch new threads and
bolts that do processing asynchronously.
Output collectors, threadsafe and can be called at anytime.
This slide explains Storm groupings and related concepts.
Part of defining a topology is specifying for
each bolt which streams it should receive as an input.
A stream grouping defines how that stream should be
partitioned among the bolt's tasks.
There are eight built-in stream groupings in Storm,
and you can define a custom stream grouping by
implementing the custom stream grouping interface.
one, shuffle grouping, tuples are randomly distributed across

the bolt's tasks in a way such that
each bolt is guaranteed to get an equal number of tuples.
Two, fields grouping, the stream is partitioned by the fields
specified in the grouping.
For example, if a stream is grouped by the user ID field,
tuples with the same user ID will always go to the same task,
but tuples with different user IDs may go to different tasks.
Three, partial key grouping,
the stream is partitioned by the fields specified in
the grouping, like the fields grouping.
But are load balanced between two downstream bolts
which provides better utilization of resources
when the incoming data is skewed.
Four, all grouping,
the stream is replicated across all the bolt's tasks.
Use this grouping with care.
Five, global grouping,
the entire stream goes to a single on of the bolt's tasks.
Specifically, it goes to the task with the lowest ID.
Six, non-grouping, this grouping specifies that you don't
care how the stream is grouped.
Currently, non-groupings are equivalent to shuffle groupings.
Eventually, though, Storm will push down bolts with
non groupings to execute in the same thread as the bolt or
spout they subscribe from, when possible.
Seven, direct grouping, this is a special kind of grouping.
A stream grouped this way means that the producer of the tuple
decides which task of the consumer will receive
this tuple.
Direct groupings can only be declared on streams that have
been declared as direct streams.
Tuples emitted to a direct stream must be emitted
using one of the methods.
A bolt can get the task IDs of its consumers by either using
the provided topology context, or by keeping track
of the output of the emit method in output collector,
which returns the task IDs that the tuple was sent to.
Finally, number eight, local or shuffle grouping.
If the target bolt has one or more tasks in the same worker
process, tuples will be shuffled to just those end process tasks.
Otherwise, this acts like a normal shuffle grouping.
Storm offers several different levels of guaranteed message
processing including best effort, at-least-once and
exactly once through Trident.
Storm guarantees every tuple will be fully processed.
One of Storms core mechanisms is the ability to truck the lineage
of a tuple as it makes its way through the topology
in an extremely efficient way.
Storm's basic abstractions provide an at-least-once
processing guarantee,

the same guarantee you get when using a queuing system.
Messages are only replayed when there are failures.
Using Trident, which is an even higher level abstraction
over Storm's basic abstractions,
you can achieve exactly once processing semantics.
Trident is a high level obstruction for
doing real-time computing on top of Storm.
It allows you to seamlessly intermix high throughput,
millions of messages per second, state for stream processing,
with low latency distributed querying.
The core data model in Trident is the stream,
processed as a series of batches.
A stream is partitioned among the nodes in the cluster and
operations applied to the stream are applied in parallel across
each partition.
There are five kinds of operations in Trident.
Operations that apply locally to each partition, and
cause no network transfer.
Repartitioning operations that repartition a stream but
otherwise don't change the contents which involves network
transfer.
Aggregation operations that do network transfer as part of
the operation, operations on grouped streams, merges and
joins.
Storm is a distributed real-time computational framework
that makes processing unbounded streams of data easy.
Trident is an abstraction on Storm,
just like Pig over Hadoop, which provides us with
various useful functions like aggregation and filter, etc.
While Storm processes streams on a per tuple basis, Trident
processes streams in batches for fault tolerant issues.
Storm provides stateless processing while Trident
provides stateful processing.
Trident provides various higher level constructs, like group by,
aggregate, joins, etc., which are built over Storm primitives,
so that we don't have to write varying logic
in different bolts to do obvious work.
With Trident, we just specify what to do with the incoming
tuples and it creates its own bolts.
Here are some pointers to state mechanisms of Storm and Trident.
Storm replays a record if not processed by a node,
processes each record at least once.
May update mutable twice.
Mutable state can be lost due to failure.
Trident uses transactions to update state.
It processes each record exactly once and
per state transaction to an external database is slow.